Compression by Switched Linear Matrix Compression

One way to think about modern neural networks is as a form of **switched linear matrix compression**.

Although a network may contain millions or billions of parameters, any particular input only activates a relatively small computational pathway. The context at each stage—whether produced by ReLU gates, attention patterns, an LSH router, or some other selection mechanism—determines **which linear mapping is used**.

Backpropagation then performs an interesting optimization. Rather than teaching a single matrix to solve every possible situation, it teaches many context-dependent matrices. Each matrix gradually becomes the **best average linear mapping** for the collection of inputs that select it.

In other words:

> *When this context is selected, this is the best linear mapping, on average, for all examples that arrive here.*

The network therefore compresses a complicated nonlinear function into a collection of relatively simple linear operators. The nonlinear behavior arises from **switching between them**, not from the matrices themselves.

Viewed this way, the compression has two components:

- **Factorization.** A large nonlinear computation is decomposed into a sequence of smaller linear transformations.
- **Context specialization.** Each transformation is only responsible for the subset of cases routed to it.

This is an elegant form of compression because each matrix only needs to model a limited region of the problem space instead of the entire function.

However, the compression is also necessarily imperfect.

Each matrix is still an **average solution** over all examples sharing the same context. Unless the context partitions the problem perfectly, every selected matrix must compromise between multiple nearby situations. Fine detail that distinguishes one example from another is effectively averaged away.

This suggests an important perspective. Rather than asking only:

> "Can we build better matrices?"

it may be equally valuable to ask:

> "Can we build better contexts?"

If the context selection becomes more informative, then each matrix is responsible for a more coherent subset of the data. The average linear mapping becomes a better approximation because there is less variation to explain within each context.

This shifts some of the burden of learning from the matrices themselves toward the routing mechanism. Better context engineering could include:

- richer routing signals,
- longer-range contextual information,
- hierarchical context descriptions,
- learned or hand-designed routing features,
- locality-sensitive hashing or other geometric partitioning methods,
- multiple interacting context sources,
- or context mechanisms that preserve useful information across many layers.

From this viewpoint, improving a neural network is not solely about increasing parameter count or matrix size. It is also about improving the quality of the partitioning of computation.

A useful mental model is that the network is continually asking:

> "Which simplified linear world does this input currently belong to?"

The better that question is answered, the simpler each local linear mapping can become.

This perspective also highlights why context-selection mechanisms deserve study in their own right. The matrices perform the local computation, but the context determines **which computation is appropriate**. As neural architectures evolve, advances in context engineering may prove just as important as advances in optimization algorithms or matrix representations.